

Kafka 中改进型 Partition 过载优化算法

颜晓莲¹, 章 刚², 邱晓红¹

(1. 江西理工大学 软件工程学院(南昌), 江西 南昌 330013;

2. 江西北大科技园, 江西 南昌 330013)

摘 要: Kafka 作为一种发布-订阅机制的高吞吐量分布式消息系统, 广受业界关注。随着应用场景垂直化、多样化, Kafka 现有的技术体系面临挑战。Partition 过载问题 (POP) 指消息分发、消息订阅等操作引起 Partition 过度服务, 并影响到物理载体 Broker 的性能。该问题是由 Kafka 中 Partition 文件配置管理的被动、僵化及孤立等不足所导致。针对此, 提出一种改进型 Partition 过载优化算法 (IPOOA)。该算法通过即时服务耗量、Partition 相似度和配置文件自动修改相结合, 实现消息分发预测以及消息分发与文件配置管理协同, 从而可有效缓解 Partition 过载问题出现。实验从 Kafka 集群 CPU 使用率、Kafka 服务延时率、Kafka 系统收敛延时比等几个方面验证了算法的有效性及其合理性。

关键词: 分布式消息系统; Kafka; Partition 过载问题; 协同管理; Broker 性能

中图分类号: TP393

文献标识码: A

文章编号: 1673-629X(2020)12-0088-04

doi: 10.3969/j.issn.1673-629X.2020.12.016

Improved Partition Overload Optimization Algorithm in Kafka

YAN Xiao-lian¹, ZHANG Gang², QIU Xiao-hong¹

(1. School of Software Engineering (Nanchang), Jiangxi University of Science and Technology,

Nanchang 330013, China;

2. Jiangxi Peking University Science Park, Nanchang 330013, China)

Abstract: Kafka, as a high-throughput distributed message system with publish subscribe mechanism, is widely concerned by the industry. Existing technology system of Kafka is facing challenges by the vertical and diversified application scenarios. Partition overload problem (POP) refers to the problem that operations such as message distribution and message subscription cause excessive service of partition which affects the performance of physical carrier broker. This problem is caused by the passivity, rigidity and isolation of partition file configuration management in Kafka. Therefore, we propose an improved optimization algorithm of partition overload (IPOOA). This algorithm combines the instant service consumption, partition similarity and automatic modification of configuration file to realize the prediction of message distribution and the coordination of message distribution and file configuration management, which can effectively alleviate the problem of partition overload. The experiment verifies the validity and rationality of the algorithm from several aspects, such as the CPU utilization rate of Kafka cluster, Kafka service delay rate, Kafka system convergence delay rate, etc.

Key words: distributed message system; Kafka; Partition overload problem; collaborative management; Broker performance

0 引言

分布式消息系统作为分布式系统重要的模块间消息传递组件, 利用可靠、高效的与平台无关的消息传递与分发, 可实现分布式系统内部解耦以及分布式系统各模块的有效集成, 因而受到业界高度关注^[1]。

ApacheKafka^[2-5]是当前较为主流、基于发布-订阅机制的高吞吐量分布式消息系统, 前期由 LinkedIn 开发后由 Apache 基金管理并开源。其优势包括: (1) 支持上层应用端多语言开发, 如 C#、JAVA、PHP、

Python、Ruby 等; (2) 支持与平台无关的消息传递与分发; (3) 支持准实时性的大规模消息处理; (4) 支持 on-line 水平扩展。相对其他消息系统, Kafka 凭借众多技术优势, 已在各行业企业级应用中普及。

在 Kafka 中, 多个 Broker (服务器) 组成 Kafka 集群, 并被 ZooKeeper 集中管理。Producer 为消息生产者, Consumer 为消息消费者, Kafka 将每个新产生的消息进行划分并归类到某个主题 Topic 中 (Topic 可理解为逻辑存储单元)。每个 Topic 被划为多个分区

收稿日期: 2019-12-26

修回日期: 2020-04-27

基金项目: 江西省教育科技项目 (GJJ170571)

作者简介: 颜晓莲 (1986-), 女, 硕士, 讲师, 研究方向为边缘计算; 通信作者: 章 刚 (1981-), 男, 博士, 研究方向为物联网、云计算。

Partition (Partition 可理解为实际存储单元), 这些分区 Partition 按某种规则均匀地部署到多个 Broker 上。根据 Kafka 系统定义和推荐, Producer 生产的消息, 依据 Hash 算法被分发至其所属 Topic 相应的 Partition 上。

Consumer 作为消费者订阅其关注的主题 Topic (可订阅多个), Kafka 按 Range 策略 (即均匀分配) 将 Consumer 分配至其关注 Topic 下众多 Partition 之一上, 由该 Partition 作为服务接入端, 并依次消费其关注的主题 Topic 下所有 Partition 的感兴趣消息。当订阅流量或分发消息数量增加时, Kafka 可通过配置文件管理增加 Partition 数量, 实现 on-line 水平扩展从而提升系统性能与吞吐量。

伴随大数据时代来临, 各行各业对大数据技术的需求越发强烈。Kafka 作为消息中间件, 不仅在分布式系统中扮演重要角色, 同时也已成为大数据流处理框架 Apache Samza 的核心组件之一。但随着 Kafka 应用的多样化, 其自身的一些不足逐渐显现。

其中不足之一便是 Partition 过载问题 (Partition overload problem, POP)。Partition 在 Kafka 中扮演承上启下角色, 上连消息生产者 Producer, 下连消息消费者 Consumer, Partition 服务性能决定着 Broker 及 Kafka 整体性能, 对 POP 问题研究将为后期优化 Broker、Consumer 乃至 Kafka 整个系统性能打下基础。

在综合衡量已有研发成果及文中所关注的重心的基础上, 认为 POP 问题指消息分发、消息存储或消息消费、消息订阅等操作造成主题 Topic 下 Partition 过度服务, 并影响到支撑 Partition 的实际物理载体 Broker 的性能。

通常而言, 在大型商业应用影响下, 某时刻会造成某个 (或多个) 主题 Topic 源源不断地涌入新消息, 并依据 Hash 算法向其 Partition 分发, 此时 Partition 不仅要处理消息存储还要处理 Consumer 的服务请求, 当新消息数量达到某阈值时, 必将导致 Partition 过载, 而这将影响到 Partition 的物理载体 Broker 的性能。

虽然, Kafka 可通过配置文件增加 Partition 数量, 缓解 Partition 过载现象出现, 但依然存在如下问题: (1) 这种由人为主观判定及人为修改的方式, 不仅准确度无法保证而且极为僵化; (2) Partition 文件配置管理与基于 Hash 算法的消息分发相互独立、相互分离, 无法根据 Partition 实际情况建立协同工作机制。这些问题的存在, 已使得 Kafka 无法满足当前多样化应用需求。

当前有关 Kafka 中 Partition 过载问题讨论极为少见。研究成果较为常见的包括: (1) ZooKeeper 集中管理机制^[6-7], 主要讨论业务复杂化后, Broker、Consumer、Consumer Group 等注册管理, Topic 与

Broker 映射关系以及 Partition 分配等管理机制, 有助于提升系统整体效率; (2) Broker^[8-9] 负载均衡, 主要讨论虚拟化背景下 Broker 如何实现接入负载均衡, 有助于提升 Broker 资源利用率; (3) Consumer^[10-11] 负载均衡, 主要讨论大规模数据处理环境下, 传统 Kafka 易造成的高开销、高误差率等问题, 有助于降低系统耗能、提升服务质量。这些虽都对 Kafka 系统实现优化, 但都无法解释 Partition 过载问题。

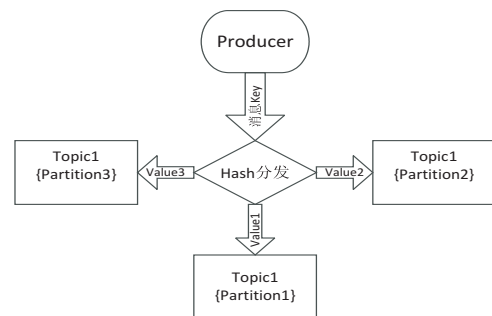
针对此, 提出一种改进型 Partition 负载优化算法 (IPOOA 算法), 该算法实现消息分发预测以及消息分发与文件配置管理协同, 从而可有效缓解 Partition 过载问题出现。

1 算法描述

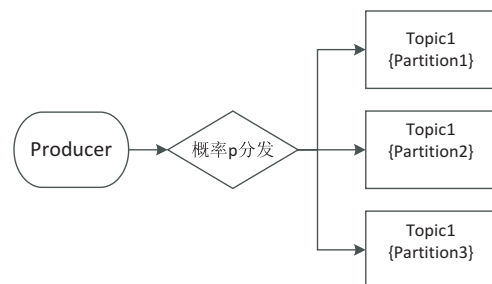
算法思想: 新消息产生后, IPOOA 算法先根据实际业务提取业务关键字 Key, 依据 Hash 分发规则计算分发至 Partition, 接着算法评估该 Partition 的即时服务耗量, 如果即时服务耗量在阈值范围内, 则新消息被分发至该 Partition, 否则算法依次计算与该 Partition 相似度较高的候选 Partition, 并评估候选 Partition 的即时服务耗量, 如果满足阈值范围, 则新消息被分发至候选 Partition, 否则重复计算候选 Partition, 直至迭代次数超过半数 Partition 总量。如果依然没有完成消息分发任务, 则通知 Kafka 自动修改配置文件新增 Partition 并存储新消息, 从而能够有效缓解 Partition 过载。

1.1 Hash 消息分发

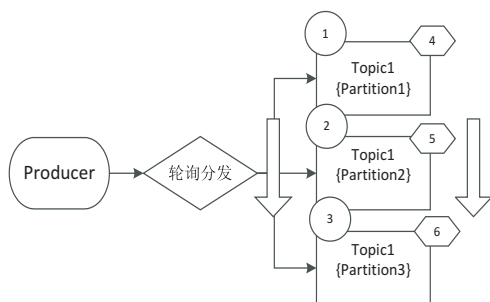
按照 Kafka 的定义, 消息分发机制共包括 Hash 分发、随机分发以及轮询分发等 (如图 1 所示), 实际中企业级应用使用范围较广的是 Hash 分发机制。



(a) Hash 分发机制



(b) 随机分发机制



(c) 轮询分发机制

图 1 各类消息分发机制示意图

该机制大致过程如下：

Step1: 指定消息的 Key (通常选取实际业务所含关键字符)；

Step2: 基于 Key 实现 Hash (Key)；

Step3: 根据 $\text{mod}(\text{Hash}(\text{key}))$ 结果将消息分发至指定 Partition；

Step4: 返回 Step1。

Hash 分发机制相对其余两种方式,其能够较好地保证消息均匀有序分发,因而被行业广泛普及使用。但 Hash 消息分发无法根据 Partition 实际负载情况进行有序分发,从而易加重 Partition 负载。

1.2 IPOOA 算法

1.2.1 即时服务耗量 (instant service consumption, ISC)

ISC 反映当前 t 时刻 Partition 中消息消费产生的服务消耗量,对任意消息 k 而言,在时刻 t 产生的服务消耗量 Cmt 由 t 时刻消息 k 订阅数 $CmtNum$ 及 t 时刻消息 k 访问连接数 $CmtCon$ 线性加权组成,如式 (1)：

$$Cmt'(k) = \lambda_1 CmtNum'(k, N_1) + \lambda_2 CmtCon'(k, N_2) \quad (1)$$

其中, $\lambda_1 \in (0, 1)$ 和 $\lambda_2 \in (0, 1)$ 为权重系数, N_1 和 N_2 分别为订阅总数和连接总数。

t 时刻 Partition 的 ISC 可表示为：

$$ISC'(Partition) = \sum_k Cmt'(k) \quad (2)$$

1.2.2 Partition 相似度 (partition similarity, PS)

PS 反映某一时刻两个 Partition 所存消息的相似程度,对任意 Partition 而言,在时刻 t 所存储消息队列表示为 $Partition^t = \{Meg_1, Meg_2, \dots, Meg_{NUM}\}$, NUM 为消息总数。则时刻 t 任意两个 Partition 的 PS 可根据加权闵可夫斯基距离 (Minkowski distance) 计算,如式 (3)：

$$\left\{ \begin{aligned} PS(Partition_A^t, Partition_B^t) &= \\ &= \left(\sum_{index=1}^{NUM} \theta_{index} |Meg_{index_A} - Meg_{index_B}|^p \right)^{\frac{1}{p}} \\ \sum_{index=1}^{NUM} \theta_{index} &= 1 \end{aligned} \right. \quad (3)$$

其中, $p \geq 1$ 为指数参数, $\theta \in (0, 1)$ 为权重系数。

1.2.3 算法过程

Step1: 初始化 Partition 配置文件, 载入 Kafka 系统中, 并设置各类参数 $\lambda_1, \lambda_2, p, \theta$ 以及 Θ (Cmt 阈值), 设定迭代次数, 转入 Step2；

Step2: 等待新消息导入, 并根据 Hash 分发算法计算其分发至 Partition, 转入 Step3；

Step3: 根据式 (1)、式 (2) 计算该 Partition 的 ISC 值, 转入 Step4；

Step4: 判定该 Partition 的 ISC 值是否满足阈值 Θ , 如果满足则新消息存储并转入 Step2.; 否则转入 Step5；

Step5: 根据式 (3) 依次计算该 Partition 与候选 Partition 的 PS 值, 并挑选出最优 PS 值, 转入 Step3；

Step6: 如果迭代次数超过半数 Partition 总量, 则通知 Kafka 自动修改配置文件新增 Partition, 并将新消息存储在新增 Partition 上, 根据实际情况转入 Step2 或转入 Step7；

Step7: 退出算法。

2 实验与分析

软硬件环境: 选取 12 个 Broker (服务器) 作为 Kafka 集群, CPU 型号为 Xeon E5-2620V3, 内存 8G, SATA 硬盘 300G, 操作系统为 SUSE Linux Enterprise Server 15。

核心参数设置: 在综合考虑文献对参数取值的建议和基于多次重复实验的结果, 参数设定如下: $p = 1$ OR $p = 2$, $\lambda_1, \lambda_2 \in (0.35, 0.65)$, $\theta_1, \theta_2, \dots \in (0.1, 0.9)$, $\Theta \in [0.5, 0.65]$, 其中实验中所有权重系数之和都为 1。

场景模拟: 12 个 Broker 服务器分成 3 个功能区, 其中 3 个服务器作为 Producer 消息生产者不断模拟分发消息, 3 个服务器作为 Consumer 消息消费者不断模拟消费消息, Producer 与 Consumer 随机分布在不同区域, 另外 6 个服务器作 Kafka 集群服务器集中管理, 处理 Producer 消息分发以及 Consumer 消息消费^[12-15]。

对比算法: 为展示实验的客观性, 分别选取传统 Kafka 算法^[2-5], 基于 Broker 负载均衡的 BL 算法^[8] 和基于 Consumer 负载均衡的 CL 算法^[10] 与融合文中 IPOOA 算法的 Kafka 相比较。

测试指标: 为体现实验的全面性, 将从多个维度验证算法的性能: (1) Kafka 集群 CPU 使用率 (Kafka CPU rate, KCR); (2) Kafka 服务延时率 (Kafka service delay rate, KSDR); (3) Kafka 系统收敛延时比 (Kafka system convergence delay rate, KSCDR)。

实验方案:

实验 1: 在并发规模为 2 000 环境下, KCR、KSDR

及 KSCDR 对比如表 1 所示。

表 1 在并发规模为 2 000 环境下,4 种算法的 KCR\KSDR\KSCDR 对比 %

算法	KCR	KSDR	KSCDR
IPOOA_Kafka	18.3	2.6	2.8
BL_Kafka	20.6	4.5	3.8
CL_Kafka	20.2	3.8	3.2
Kafka	22.8	5.1	4.3

实验 2:在并发规模为 3 500 环境下,KCR、KSDR 及 KSCDR 对比如表 2 所示。

表 2 在并发规模为 3 500 环境下,4 种算法的 KCR\KSDR\KSCDR 对比 %

算法	KCR	KSDR	KSCDR
IPOOA_Kafka	27.7	4.3	4.9
BL_Kafka	32.9	6.7	7.5
CL_Kafka	33.2	7.0	8.6
Kafka	39.3	9.2	10.1

实验 3:在并发规模为 5 000 环境下,KCR、KSDR 及 KSCDR 对比如表 3 所示。

表 3 在并发规模为 5 000 环境下,4 种算法的 KCR\KSDR\KSCDR 对比 %

算法	KCR	KSDR	KSCDR
IPOOA_Kafka	38.6	9.3	9.8
BL_Kafka	43.6	12.5	13.2
CL_Kafka	45.5	13.1	13.9
Kafka	51.2	20.9	22.2

实验总结:在并发规模逐渐增加下,融合文中算法的 Kafka 系统(IPOOA_Kafka)在各项指标层面相对较优,主要原因在于 IPOOA_Kafka 能够实现预测消息分发以及消息分发与文件配置协同工作,从而能缓解 Partition 过载问题出现,提升系统整体性能。

3 结束语

针对 Kafka 中 Partition 文件配置管理所存在的被动、僵化及孤立等不足,使得 Partition 过载问题无法有效解决,提出一种改进型 Partition 过载优化算法。该算法通过即时服务耗量,Partition 相似度和配置文件自动修改相结合,实现消息分发预测以及消息分发与文件配置管理协同,从而可有效缓解 Partition 过载问题出现。实验从 Kafka 集群 CPU 使用率、Kafka 服务延时率、Kafka 系统收敛延时比等几个方面验证了算法的有效性及其合理性。未来将重点围绕消息分发、消息订阅及文件配置管理等多层面协同展开研究。

参考文献:

- [1] 吴 臻,王小宁,肖海力,等. 分布式消息系统研究综述[J]. 计算机科学,2019,46(21):1-5.
- [2] PROULX T, HEINE S J. Connections from Kafka: exposure to meaning threats improves implicit learning of an artificial grammar[J]. Psychological Science, 2009, 20(9): 1125-1131.
- [3] 于晓鹏,陈建孝,李永丽. 基于消息队列的分布式系统数据一致性方法研究[J]. 吉林大学学报:信息科学版,2011,29(3):258-262.
- [4] 易 佳,薛 晨,王树鹏. 分布式流数据加载和查询技术优化[J]. 计算机科学,2017,44(5):172-177.
- [5] 金双喜,李 永,吴 骅,等. 基于 Kafka 消息队列的新一代分布式电量采集方法研究[J]. 智慧电力,2018,46(2):77-82.
- [6] 孙龙龙. 面向 Kafka 消息中间件的监控管理系统设计与实现[D]. 武汉:华中科技大学,2018.
- [7] 王宝童. Kafka 多集群管理系统的设计与实现[D]. 武汉:华中科技大学,2017.
- [8] YI Meng, CHEN Qingkui, ZHANG Gang. Multistage dynamic packet access mechanism of internet of things[J]. Mobile Information Systems, 2018, 2018(6): 3601604. 1-3601604. 16.
- [9] 王志泳. 分布式消息系统 Kafka 的性能建模与优化技术研究[D]. 西安:西安电子科技大学,2017.
- [10] 王郑合,王 锋,邓 辉,等. 一种优化的 Kafka 消费者/客户端负载均衡算法[J]. 计算机应用研究,2017,34(8):2306-2309.
- [11] 蒋海波. 海量数据存储系统的高可靠性关键技术研究与应用[D]. 成都:电子科技大学,2013.
- [12] GUO Bingli, SHANG Yu, ZHANG Yunquan. Timeslot switching-based optical bypass in data center for Intrarack elephant flow with an ultrafast DPDK-enabled timeslot allocator[J]. Journal of Lightwave Technology, 2019, 37(10): 2253-2260.
- [13] WANG Guozhang, JOEL K. Building a replicated logging system with Apache Kafka[J]. Proceedings of the VLDB Endowment, 2015, 8(12): 1654-1656.
- [14] WISKA R, HABIBIE N. Big sensor-generated data streaming using Kafka and impala for data storage in wireless sensor network for CO2 monitoring[C]//2016 international workshop on big data and information security. Jakarta: IEEE, 2017: 97-102.
- [15] HUNKELER U, TRUONG H L, STANFORD-CLARK A. MQTT-S - A publish/subscribe protocol for wireless sensor networks[C]//2008 3rd international conference on communication systems software and middleware and workshops. Bangalore: IEEE, 2008: 791-798.